16 | 内存篇：如何减少延迟提升内存分配效率？

2020-01-01 庄振运

性能工程高手课 进入课程 >

讲述：庄振运
时长 16:43　大小 13.40M

你好，我是庄振运。

上一讲我们讨论了关于 CPU 的性能指标和分析。CPU 和内存是和程序性能最相关的两个领域；那么这一讲，我们就来讨论和内存相关的性能指标和性能分析的工具。

内存方面的性能指标，主要有缓存命中率、缓存一致性、内存带宽、内存延迟、内存的使用大小及碎片、内存的分配和回收速度等，接下来我会逐一进行介绍。现代很多 CPU 都是 NUMA 架构的，所以我也会介绍 NUMA 的影响和常用的工具。

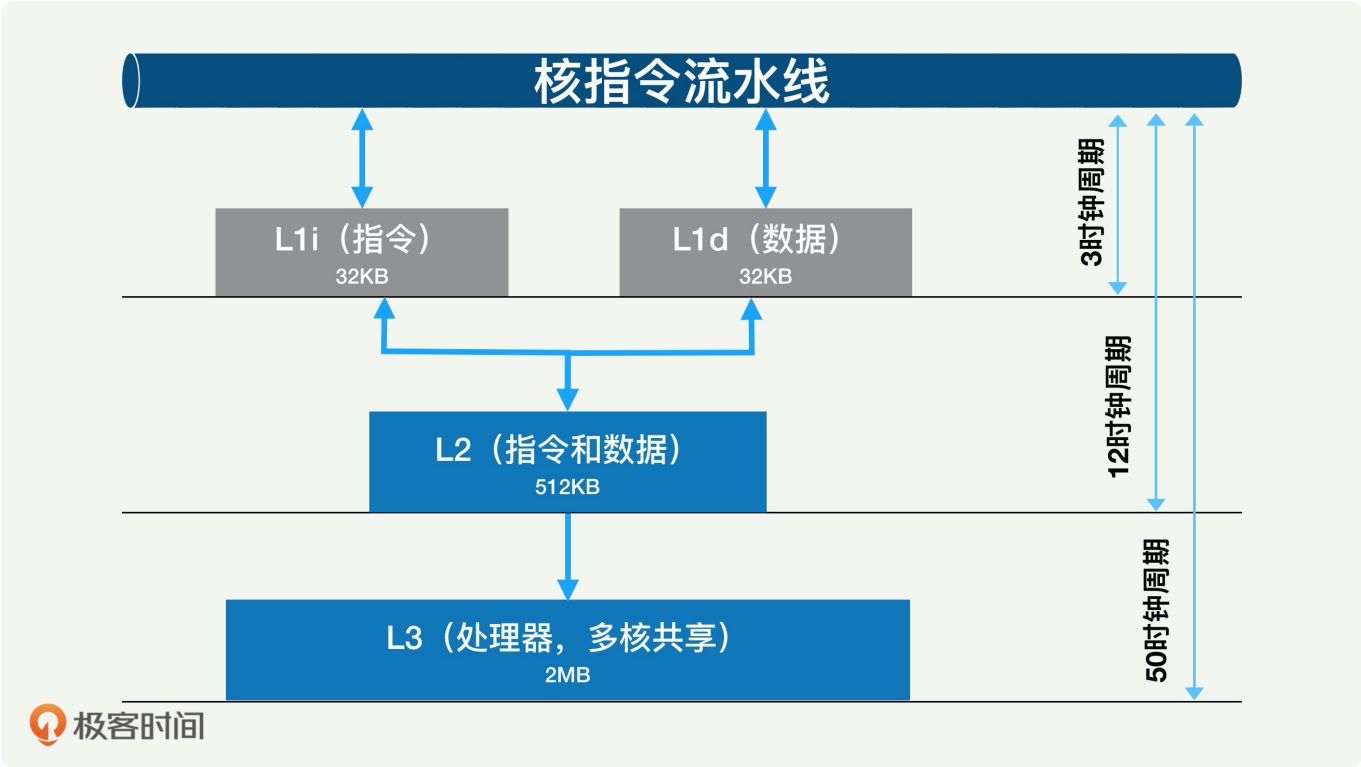
缓存和缓存命中率

我们先看看缓存，也就是 Cache。

缓存是 CPU 与内存之间的临时数据交换器，是为了解决两种速度不匹配的矛盾而设计的。这个矛盾就是 **CPU 运行处理速度**与**内存读写速度**不匹配的矛盾。CPU 处理指令的速度，比内存的速度快得多了，有百倍的差别，这一点我们已经在上一讲讨论过。

缓存的概念极为重要。不止是 CPU，缓存的策略也用在计算机和互联网服务中很多其他的地方，比如外部存储、文件系统，以及程序设计上。有人甚至开玩笑说，计算机的各种技术说到底就是三种——Cache（缓存）、Hash（哈希处理）和 Trash（资源回收）。这种说法当然有点偏颇，但你也能从中看到缓存技术的重要性。

现在回到 CPU 缓存的讨论上来。



我们前面也讲了，随着多核 CPU 的发展，CPU 缓存通常分成了三个级别：L1、L2、L3。一般而言，每个核上都有 L1 和 L2 缓存。L1 缓存其实分成两部分：一个用于存数据，也就是 L1d Cache（Data Cache），另外一个用于存指令，L1i Cache（Instruction Cache）。

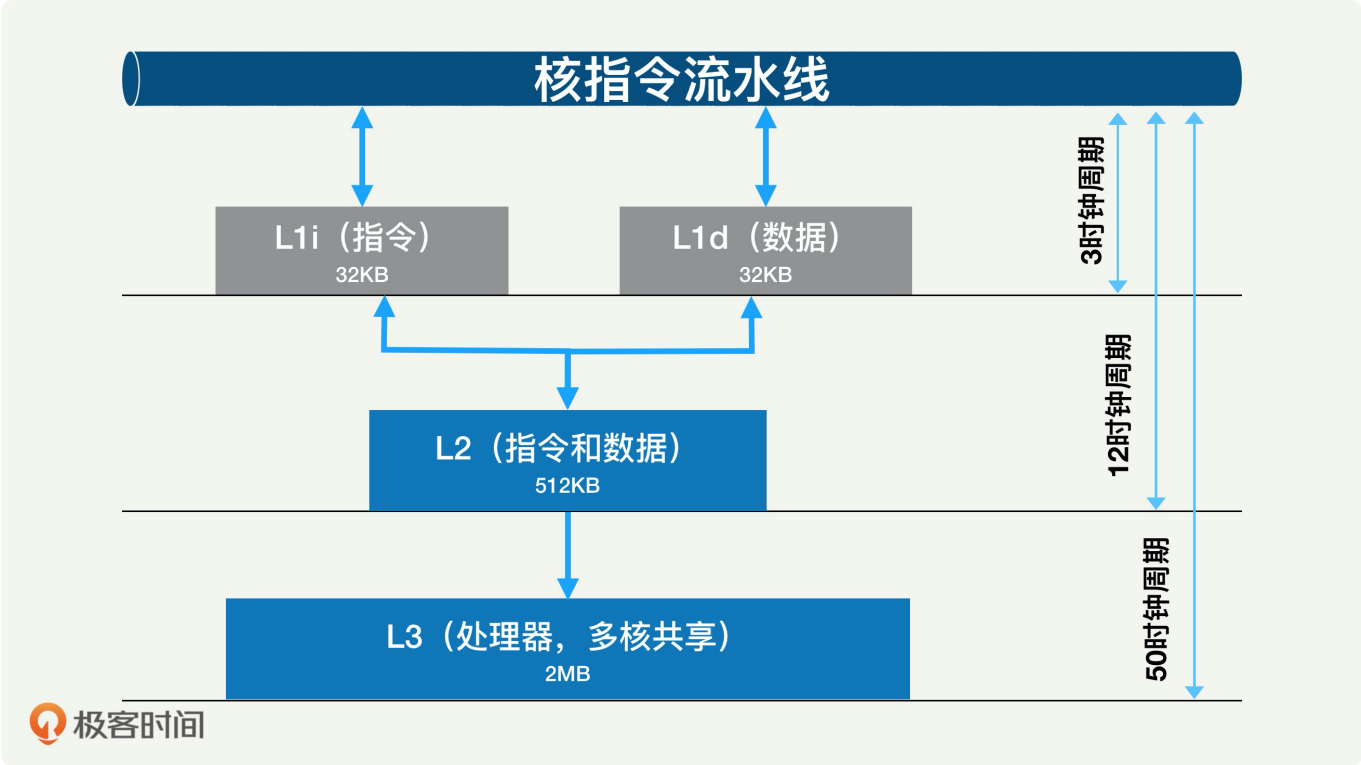
L1 缓存相对较小，每部分差不多只有几十 KB。L2 缓存更大一些，有几百 KB，速度就要慢一些了。L2 一般是一个统一的缓存，不把数据和指令分开。L3 缓存则是三级缓存中最大的

一级，可以达到几个 MB 大小，同时也是最慢的一级了。你要注意，在同一个处理器上，所有核共享一个 L3 缓存。

为什么要采用多级缓存，并逐级增加缓存大小呢？

这个目的，就是为了提高各级缓存的命中率，从而最大限度地降低直接访问内存的概率。每一级缓存的命中率都很重要，尤其是 L1 的命中率。这是因为缓存的命中率对总体的访问时间延迟影响很大，而且下一级缓存的访问延迟往往是上一级缓存延迟的很多倍。

为了加深你的理解，我还是用文章中图片里的延迟数据来举例说明一下。



在图片里你可以看到，L1 的访问时间是 3 个时钟周期，L2 的访问时间是 12 个时钟周期。假如在理想情况下，L1i 的命中率是 100%，就是说每条指令都从 L1i 里面取；那么平均指令访问时间也是 3 个时钟周期。作为对比，如果 L1i 命中率变成 90%，也就是只有 90% 的指令从 L1i 里面取到，而剩下的 10% 需要 L2 来提供。

那么平均指令访问时间就变成了 3.9 个指令周期（也就是：90%*3+10%*12）。虽然看起来只有 10% 的指令没有命中，但是相对于 L1 命中率 100% 的情况，平均访问时间延迟差不多增大了多少呢？高达 30%。

缓存一致性

虽然缓存能够极大地提升运算性能，但也带来了一些其他的问题，比如“缓存一致性问题（cache coherence）”。

如果不能保证缓存一致性，就可能造成结果错误。因为每个核都有自己的 L1 和 L2 缓存，当在不同核上运行同一个进程的不同线程时，如果这些线程同时操作同一个进程内存，就可能互相冲突，最终产生错误的结果。

举个例子，你可以设想这样一个场景，假设处理器有两个核，core-A 和 core-B。这两个核同时运行两个线程，都操作共同的变量，i。假设它们并行执行 i++。如果 i 的初始值是 0，当两个线程执行完毕后，我们预期的结果是 i 变成 2。但是，如果不采取必要的措施，那么在实际执行中就可能会出错，什么样的错误呢？我们来探讨一下。

运行开始时，每个核都存储了 i 的值 0。当第 core-A 做 i++ 的时候，其缓存中的值变成了 1，i 需要马上回写到内存，内存之中的 i 也就变成了 1。但是 core-B 缓存中的 i 值依然是 0，当运行在它上面的线程执行 i++，然后回写到内存时，就会覆盖 core-A 内核的操作，使得最终 i 的结果是 1，而不是预期中的 2。

为了达到数据访问的一致，就需要各个处理器和内核，在访问缓存和写回内存时遵循一些协议，这样的协议就叫**缓存一致性协议**。常见的缓存一致性协议有 MSI、MESI 等。

缓存一致性协议解决了缓存内容不一致的问题，但同时也造成了缓存性能的下降。在有些情况下性能还会受到严重影响。我们下一讲还会仔细分析这一点，并且讨论怎样通过优化代码来克服这样的性能问题。

内存带宽和延迟

我们讨论了缓存，接下来探讨内存带宽和内存访问延迟。

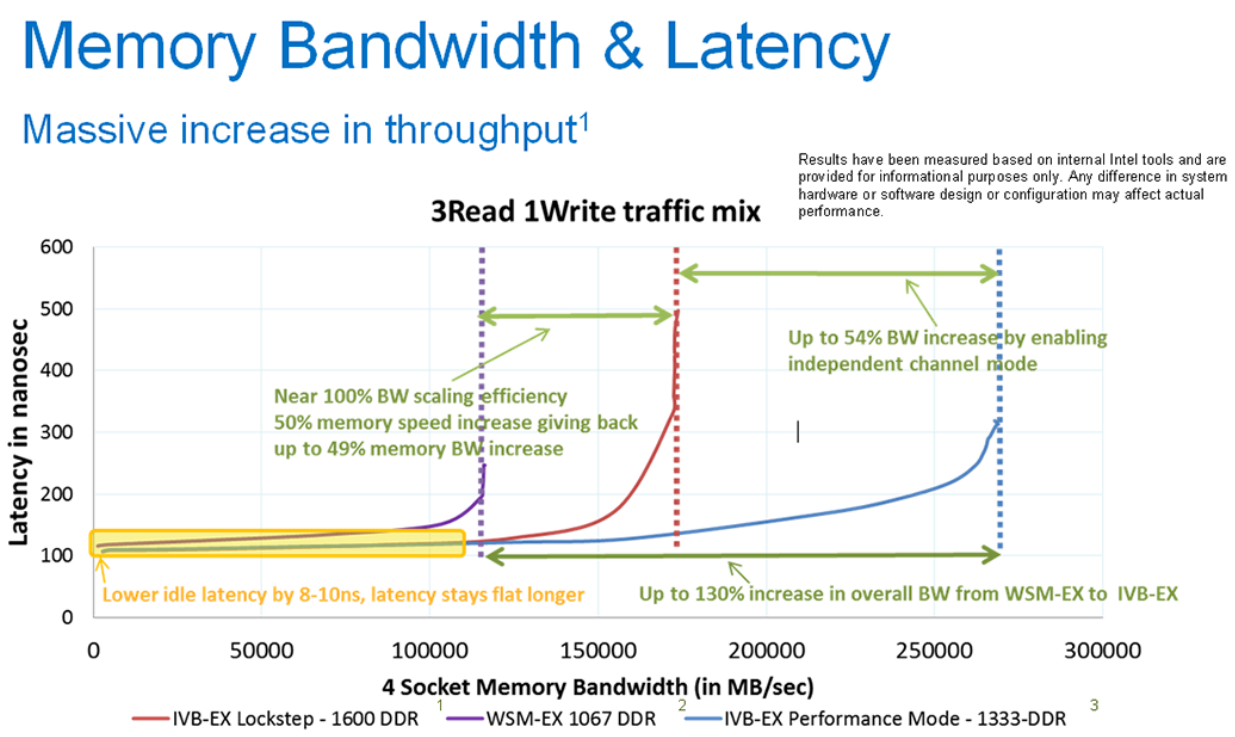
计算机性能方面的一个趋势就是，内存越来越变成主要的性能瓶颈。内存对性能的制约包括三个方面：内存大小、内存访问延迟和内存带宽。

第一个方面就是内存的使用大小，这个最直观，大家都懂。这方面的优化方式也有很多，包括采用高效的，使用内存少的算法和数据结构。

第二个方面是内存访问延迟，这个也比较好理解，我们刚刚讨论的各级缓存，都是为了降低内存的直接访问，从而间接地降低内存访问延迟的。如果我们尽量降低数据和程序的大小，那么各级缓存的命中率也会相应地提高，这是因为缓存可以覆盖的代码和数据比例会增大。

第三个方面就是内存带宽，也就是单位时间内，可以并行读取或写入内存的数据量，通常以字节 / 秒为单位表示。一款 CPU 的最大内存带宽往往是有限而确定的。并且一般来说，这个最大内存带宽只是个理论最大值，实际中我们的程序使用只能达到最大带宽利用率的 60%。如果超出这个百分比，内存的访问延迟会急剧上升。

文章中的图片就展示了几款 Intel 的 CPU 的内存访问延迟和内存带宽的关系（图片来自 https://images.anandtech.com）。



图中内存带宽使用是横轴，相对应的，内存访问延迟是纵轴。你可以清楚地看到，当内存带宽较小时，内存访问延迟很小，而且基本固定，最多缓慢上升。而当内存带宽超过一定值后，访问延迟会快速上升，最终增加到不能接受的程度。

那么一款处理器的内存总带宽取决于哪些因素呢？

答案是，有四个因素，内存总带宽的大小就是这些因素的乘积。这四个因素是：DRAM 时钟频率、每时钟的数据传输次数、内存总线带宽（一般是 64 字节）、内存通道数量。

我们来用几个实际的 Intel CPU 为例，来看看内存带宽的变化。

	SandyBridge	IvyBridge	Haswell (1P)	Haswell (2P)	Broadwell (1P)	Broadwell (2P)	Skylake (1P)	Skylake (2P)
L1 Cache (per core)	32KB I&D	32KB I&D	32KB I&D	32KB I&D	32KB I&D	32KB I&D	32KB I&D	32KB I&D
L2 Cache (per core)	256KB	256KB	256KB	256KB	256KB	256KB	1MB	1MB
L3 Cache(per socket)	20MB	25MB	30MB	30MB	20MB	35MB	24MB	27.5MB
Memory DRAM通道	4	4	4	8	2	8	4	8
内存带宽 (50%)	15 GB/sec	22 GB/sec	38 GB/sec	60 GB/sec	21 GB/sec	76 GB/sec	38 GB/s	85 GBs
内存频率和速度	DDR3 1333MHz	DDR3 1600MHz	DDR3 1866MHz	DDR3 1600MHz	DDR4 2133MHz	DDR4 2400MHz	DDR 2400 MHz	DDR 2667 MHz

文章中的这个表格大体上总结了 5 款 Intel CPU 的各级缓存大小、内存通道数目、可使用内存带宽（这里取最大值的 50%）、内存频率和速度。你可以看到，每款新的 CPU，它的内存带宽一般还是增加的，这主要归功于内存频率的提升。

内存的分配

讲过内存带宽，我们再来看看内存的分配。程序使用的内存大小很关键，是影响一个程序性能的重要因素，所以我们应该尽量对程序的内存使用大小进行调优，从而让程序尽量少地使用内存。

不知道你有没有过系统内存用光的经验？每当发生这种情况，系统就会被迫杀掉一些进程，并且抛出一个系统错误：内存用光“OOM（Out of memory）”。所以，一个应用程序用的内存越少，那么 OOM 错误就越不太可能发生。

还有，服务器等容量也是公司运营成本的一部分。如果一台服务器的内存资源足够，那么这样一个服务器系统就可以同时运行多个程序或进程，以最大限度地提高系统利用率，这样就节省了公司运营成本。

再进一步讲，应用程序向操作系统申请内存时，系统会分配内存，这中间总要花些时间，因为操作系统需要查看可用内存并分配。一个系统的空闲内存越多，应用程序向操作系统申请

内存的时候，就越快地拿到所申请的内存。反之，应用程序就有可能经历很大的内存请求分配延迟。

比如说，在系统空闲内存很少的时候，程序很可能会变得超级慢。因为操作系统对内存请求进行（比如 malloc()）处理时，如果空闲内存不够，系统需要采取措施回收内存，这个过程可能会阻塞。

我们写程序时，或许习惯直接使用 new、malloc 等 API 申请分配内存，直观又方便。但这样做有个很大的缺点，就是所申请内存块的大小不定。当这样的内存申请频繁操作时，会造成大量的内存碎片；这些内存碎片会导致系统性能下降。

一般来讲，开发应用程序时，采用内存池（Memory Pool）可以看作是一种内存分配方式的优化。

所谓的内存池，就是提前申请分配一定数量的、大小仔细考虑的内存块留作备用。当线程有新的内存需求时，就从内存池中分出一部分内存块。如果已分配的内存块不够，那么可以继续申请新的内存块。同样，线程释放的内存也暂时不返还给操作系统，而是放在内存池内留着备用。

这样做的一个显著优点是尽量避免了内存碎片，使得内存分配效率和系统的总体内存使用效率得到提升。

NUMA 的影响

我们刚刚谈了内存性能的几个方面，最后看看多处理器使用内存的情景，也就是 NUMA 场景。NUMA 系统现在非常普遍，它和 CPU 和内存的性能都很相关。简单来说，NUMA 包含多个处理器（或者节点），它们之间通过高速互连网络连接而成。每个处理器都有自己的本地内存，但所有处理器可以访问全部内存。

因为访问远端内存的延迟远远大于本地内存访问，操作系统的设计已经将内存分布的特点考虑进去了。比如一个线程运行在一个处理器中，那么为这个线程所分配的内存，一般是该处理器的本地内存，而不是外部内存。但是，在特殊情况下，比如本地内存已经用光，那就只能分配远端内存。

我们部署应用程序时，最好将访问相同数据的多个线程放在相同的处理器上。根据情况，有时候也需要强制去绑定线程到某个节点或者 CPU 核上。

工具

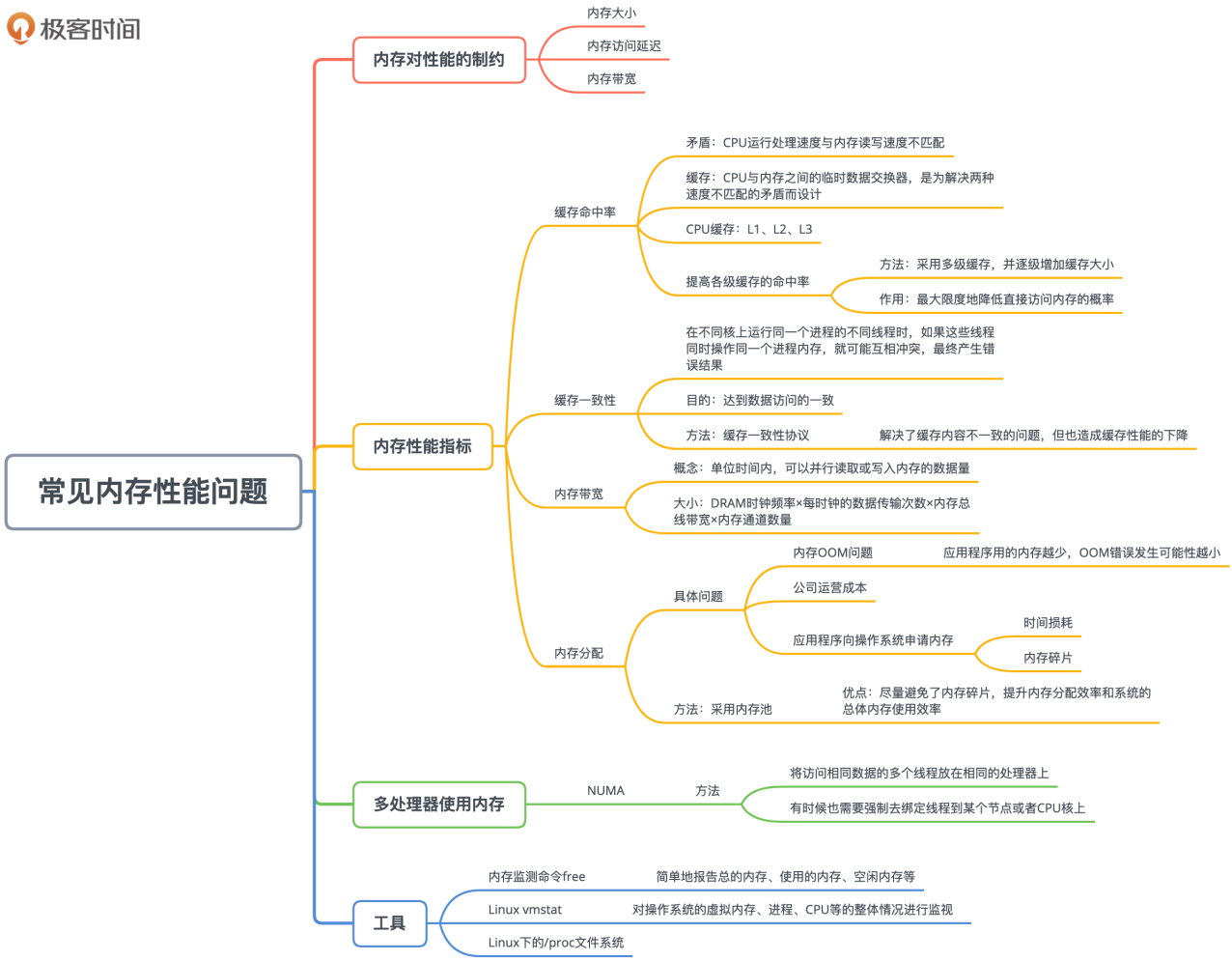
内存相关的工具也挺多的。比如，你最熟的内存监测命令或许是 free 了。这个命令会简单地报告总的内存、使用的内存、空闲内存等。

vmstat（Virtual Meomory Statistics， 虚拟内存统计）也是 Linux 中监控内存的常用工具，可以对操作系统的虚拟内存、进程、CPU 等的整体情况进行监视。

我建议你也尽量熟悉一下 Linux 下的 /proc 文件系统。这是一个虚拟文件系统，只存在内存当中，而不占用外存空间。这个目录下有很多文件，每一个文件的内容都是动态创建的。这些文件提供了一种在 Linux 内核空间和用户间之间进行通信的方法。比如 /proc/meminfo 就对内存方面的监测非常有用。这个文件里面有几十个条目，比如 SwapFree，显示的是空闲 swap 总量等。

另外，/proc 这个目录下还可以根据进程的 ID 来查看每个进程的详细信息，包括分配到进程的内存使用。比如 /proc/PID/maps 文件，里面的每一行都描述进程或线程中连续虚拟内存的区域；这些信息提供了更深层次的内存剖析。

总结



我们都知道宋代词人辛弃疾，他曾经这样憧憬他的战场梦想：“马作的卢飞快，弓如霹雳弦惊。” 我们开发的应用程序对内存的分配请求延迟，也有相似的期盼，就是要动作飞快。如果内存分配延迟太大，整个程序的性能自然也高不上去。

如何实现这个梦想呢？就需要我们的代码和程序，尽量降低对内存的使用大小和内存带宽，尽量少地请求分配和释放内存，帮助系统内存状态不至于太过碎片化，并且对代码结构做一些相应地优化。

思考题

你正在开发的系统或者模块，会运行在什么样的服务器上？服务器上有多少内存？如果内存大小可能不够，你会采取什么措施来降低内存使用量呢？再进一步，内存带宽会是瓶颈吗？

欢迎你在留言区分享自己的思考，与我和其他同学一起讨论，也欢迎你把文章分享给自己的朋友。

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇　15 | CPU篇：如何让CPU的运行不受阻碍?

下一篇　17 | 存储篇：如何根据性能优缺点选择最合适的存储系统?

精选留言 (2)

写留言

Enzo
2020-01-09

malloc 函数已经实现类似内存池的功能，在应用程序中再实现一个内存池封装，会有性能提升吗?

> 作者回复: malloc本身应该是最基本的内存申请功能，你是说jemalloc吧？那个的确有帮助；但是jemalloc也有很多问题。这里的“内存池”偏重在概念，和具体的实现分开。

Enzo
2020-01-09

SIMD指令可以显著提高缓存命中率，大量数据处理的时候用的多